Deep, Shallow and Lazy Copy with Java Examples

Difficulty Level : Medium • Last Updated : 13 Jun, 2022

In object-oriented programming, object copying is creating a copy of an existing object, the resulting object is called an object copy or simply copy of the original object.There are several ways to copy an object, most commonly by a copy constructor or cloning.

We can define Cloning as **"create a copy of object".** Shallow, deep and lazy copy is related to cloning process.

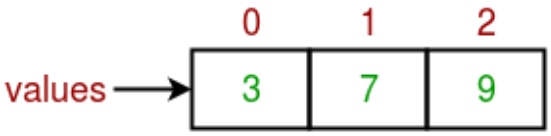
These are actually three ways for creating copy object.

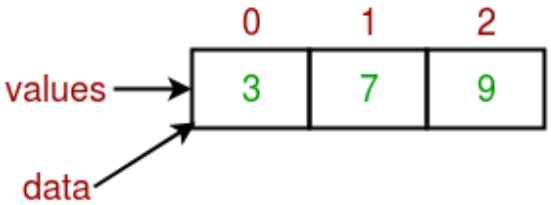
Shallow Copy

- Whenever we use default implementation of clone method we get shallow copy of object means it creates new instance and copies all the field of object to that new instance and returns it as object type, we need to explicitly cast it back to our original object. This is shallow copy of the object.
- clone() method of the object class support shallow copy of the object. If the object contains primitive as well as non primitive or reference type variable in shallow copy, the cloned object also refers to the same object to which the original object refers as only the object references gets copied and not the referred objects themselves.
- That's why the name shallow copy or shallow cloning in Java. If only primitive type fields or Immutable objects are there then there is no difference between shallow and deep copy in Java.

```java
//code illustrating shallow copy
public class Ex {

    private int[] data;

    // makes a shallow copy of values
    public Ex(int[] values) {
        data = values;
    }

    public void showData() {
        System.out.println( Arrays.toString(data) );
    }
}
```

The above code shows shallow copying. data simply refers to the same array as vals.





Shallow Copy

This can lead to unpleasant side effects if the elements of values are changed via some other reference.

Java

```
public class UsesEx{

    public static void main(String[] args) {
        int[] vals = {3, 7, 9};
        Ex e = new Ex(vals);
        e.showData(); // prints out [3, 7, 9]
        vals[0] = 13;
        e.showData(); // prints out [13, 7, 9]

        // Very confusing, because we didn't
        // intentionally change anything about
        // the object e refers to.
    }
}
```

```
Output 1 : [3, 7, 9]
Output 2 : [13, 7, 9]
```

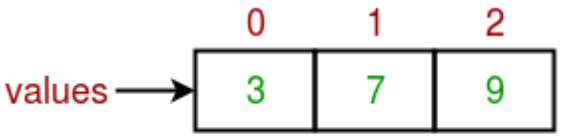
Deep Copy

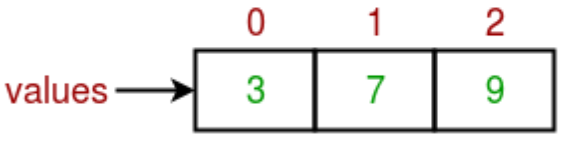
- Whenever we need own copy not to use default implementation we call it as deep copy, whenever we need deep copy of the object we need to implement according to our need.

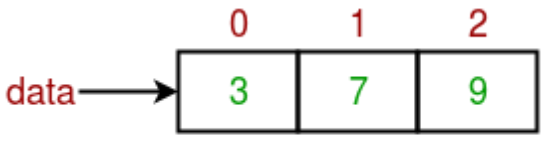
A deep copy means actually creating a new array and copying over the values.

Java

```
// Code explaining deep copy
public class Ex {

    private int[] data;

    // altered to make a deep copy of values
    public Ex(int[] values) {
        data = new int[values.length];
        for (int i = 0; i < data.length; i++) {
            data[i] = values[i];
        }
    }

    public void showData() {
        System.out.println(Arrays.toString(data));
    }
}
```







Deep Copy

```
public class UsesEx{

    public static void main(String[] args) {
        int[] vals = {3, 7, 9};
        Ex e = new Ex(vals);
        e.showData(); // prints out [3, 7, 9]
        vals[0] = 13;
        e.showData(); // prints out [3, 7, 9]

       // changes in array values will not be
       // shown in data values.
    }
}
```

```
Output 1 : [3, 7, 9]
Output 2 : [3, 7, 9]
```

Changes to the array vals will not result in changes to the array data.

when to use what

There is no hard and fast rule defined for selecting between shallow copy and deep copy but normally we should keep in mind that if an object has only primitive fields, then obviously we should go for shallow copy, but if the object has references to other objects, then based on the requirement, shallow copy or deep copy should be done. If the references are not updated then there is no point to initiate a deep copy.

Lazy Copy

A lazy copy can be defined as a combination of both shallow copy and deep copy. The mechanism follows a simple approach – at the initial state, shallow copy approach is used. A counter is also used to keep a track on how many objects share the data. When the program wants to modify the original object, it checks whether the object is shared or not. If the object is shared, then the deep copy mechanism is initiated.

Summary

In shallow copy, only fields of primitive data type are copied while the objects references are not copied. Deep copy involves the copy of primitive data type as well as object references. There is no hard and fast rule as to when to do shallow copy and when to do a deep copy. Lazy copy is a combination of both of these approaches

Start Your Coding Journey Now!

1. Java Program to Show Shallow Cloning and Deep Cloning
2. Java Program to Demonstrate the Non-Lazy Initialization Thread-Safe
3. Java Program to Demonstrate the Lazy Initialization Thread-Safe
4. Java Program to Demonstrate the Lazy Initialization Non-Thread-Safe
5. How to Lazy Load Images in Android ListView?
6. Hibernate - Eager/Lazy Loading
7. Collections copy() method in Java with Examples
8. Files copy() Method in Java with Examples
9. Deep Linking in Android with Example
10. Copy Elements of One Java Vector to Another Vector in Java

Previous Next

Article Contributed By :

GeeksforGeeks

Start Your Coding Journey Now!

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : gabaa406, pratyaksh16

Article Tags : Java

Practice Tags : Java

Improve Article | Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org